Topics in TCS

Majority and Misra-Gries

Raphaël Clifford

The majority problem

Given a sequence of integers $a_1, \ldots, a_m$, does there exist a integer that occurs more than $m/2$ times?

The majority problem

Given a sequence of integers $a_1, \ldots, a_m$, does there exist a integer that occurs more than $m/2$ times?

Originally considered for elections. Three candidates A, B and C. Did any of them get a majority?

The majority problem

Given a sequence of integers $a_1, \ldots, a_m$, does there exist a integer that occurs more than $m/2$ times?

Originally considered for elections. Three candidates A, B and C. Did any of them get a majority?

Naive majority solution

Votes: $AAACCBBCCCBCC$. There were 13 votes in total.

The majority problem

Given a sequence of integers $a_1, \ldots, a_m$, does there exist a integer that occurs more than $m/2$ times?

Originally considered for elections. Three candidates A, B and C. Did any of them get a majority?

Naive majority solution

Votes: $AAACCBBCCCBCC$. There were 13 votes in total.

We could sort the input giving $AAABBBCCCCCCC$.

The majority problem

Given a sequence of integers $a_1, \ldots, a_m$, does there exist a integer that occurs more than $m/2$ times?

Originally considered for elections. Three candidates A, B and C. Did any of them get a majority?

Naive majority solution

Votes: $AAACCBBCCCBCC$. There were 13 votes in total.

We could sort the input giving $AAABBBCCCCCCC$.

Then traverse in linear time to find if any occur ≥ 7 times.

The majority problem

Given a sequence of integers $a_1, \ldots, a_m$, does there exist a integer that occurs more than $m/2$ times?

Originally considered for elections. Three candidates A, B and C. Did any of them get a majority?

Naive majority solution

Votes: $AAACCBBCCCBCC$. There were 13 votes in total.

We could sort the input giving $AAABBBCCCCCCC$.

Then traverse in linear time to find if any occur ≥ 7 times.

Linear space and $O(m \log m)$ time.

Finding the majority

Solved in 1981 by Boyer and Moore when considering votes. Run MAJORITY for each item in the input.

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
  if c == 0
        α = j
        c = 1
  elif j == α
        c = c + 1
  else
        c = c - 1
```

Majority algorithm

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
      if c == 0
          α = j
          c = 1
      elif j == α
          c = c + 1
      else
          c = c - 1
```

Consider the stream arriving from left to right.

$$\underline{A}AACCBBCCCBCC$$

$\alpha = \text{A}$
c $= 1$

Majority algorithm

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
        if c == 0
            α = j
            c = 1
        elif j == α
            c = c + 1
        else
            c = c - 1
```

Consider the stream arriving from left to right.

$$A\underline{A}ACCBBCCCBCC$$

$\alpha = \mathsf{A}$
c $= 2$

Majority algorithm

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
        if c == 0
            α = j
            c = 1
        elif j == α
            c = c + 1
        else
            c = c - 1
```

Consider the stream arriving from left to right.

$$AA\underline{A}CCBBCCCBCC$$

$\alpha = \text{A}$
c $= 3$

Majority algorithm

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
      if c == 0
           α = j
           c = 1
      elif j == α
           c = c + 1
      else
           c = c - 1
```

Consider the stream arriving from left to right.

$$AAA\underline{C}CBBCCCBCC$$

$\alpha = \text{A}$
c $= 2$

Majority algorithm

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
      if c == 0
         α = j
         c = 1
      elif j == α
         c = c + 1
      else
         c = c - 1
```

Consider the stream arriving from left to right.

$$AAAC\underline{C}BBCCCBCC$$

$\alpha = \text{A}$
c $= 1$

Majority algorithm

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
        if c == 0
            α = j
            c = 1
        elif j == α
            c = c + 1
        else
            c = c - 1
```

Consider the stream arriving from left to right.

$$AAACC\underline{B}BCCCBCC$$

$\alpha = \mathrm{A}$
c $= 0$

Majority algorithm

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
        if c == 0
            α = j
            c = 1
        elif j == α
            c = c + 1
        else
            c = c - 1
```

Consider the stream arriving from left to right.

$$AAACCB\underline{B}CCCBCC$$

$\alpha = \mathrm{B}$
c $= 1$

Majority algorithm

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
       if c == 0
           α = j
           c = 1
       elif j == α
           c = c + 1
       else
           c = c - 1
```

Consider the stream arriving from left to right.

$$AAACCBB\underline{C}CCBCC$$

$\alpha = \mathrm{B}$
c $= 0$

Majority algorithm

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
      if c == 0
          α = j
          c = 1
      elif j == α
          c = c + 1
      else
          c = c - 1
```

Consider the stream arriving from left to right.

$$AAACCBBC\underline{C}CBCC$$

$\alpha = \mathrm{C}$
c $= 1$

Majority algorithm

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
        if c == 0
            α = j
            c = 1
        elif j == α
            c = c + 1
        else
            c = c - 1
```

Consider the stream arriving from left to right.

$$AAACCBBCC\underline{C}BCC$$

$\alpha = \mathsf{C}$
c $= 2$

Majority algorithm

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
        if c == 0
            α = j
            c = 1
        elif j == α
            c = c + 1
        else
            c = c - 1
```

Consider the stream arriving from left to right.

$$AAACCBBCCC\underline{B}CC$$

$\alpha = \mathrm{C}$
c $= 1$

Majority algorithm

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
      if c == 0
          α = j
          c = 1
      elif j == α
          c = c + 1
      else
          c = c - 1
```

Consider the stream arriving from left to right.

$$AAACCBBCCCB\underline{C}C$$

$\alpha = \mathrm{C}$

$\mathrm{c} = 2$

Majority algorithm

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
        if c == 0
            α = j
            c = 1
        elif j == α
            c = c + 1
        else
            c = c - 1
```

Consider the stream arriving from left to right.

$$AAACCBBCCCBC\underline{C}$$

$\alpha = \mathrm{C}$
c $= 3$

C is the majority item ✓

Majority algorithm - wrong answer

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
      if c == 0
            α = j
            c = 1
      elif j == α
            c = c + 1
      else
            c = c - 1
```

Consider the stream arriving from left to right.

$$\underline{A}AABBBC$$

$\alpha = \text{A}$

$\text{c} = 1$

Majority algorithm - wrong answer

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
      if c == 0
          α = j
          c = 1
      elif j == α
          c = c + 1
      else
          c = c - 1
```

Consider the stream arriving from left to right.

$$A\underline{A}ABBBC$$

$\alpha = \text{A}$
c $= 2$

Majority algorithm - wrong answer

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
        if c == 0
            α = j
            c = 1
        elif j == α
            c = c + 1
        else
            c = c - 1
```

Consider the stream arriving from left to right.

$$AA\underline{A}BBBC$$

$\alpha = \mathrm{A}$
c $= 3$

Majority algorithm - wrong answer

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
      if c == 0
           α = j
           c = 1
      elif j == α
           c = c + 1
      else
           c = c - 1
```

Consider the stream arriving from left to right.

$$AAA\underline{B}BBC$$

$\alpha = \text{A}$

$\text{c} = 2$

Majority algorithm - wrong answer

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
      if c == 0
           α = j
           c = 1
      elif j == α
           c = c + 1
      else
           c = c - 1
```

Consider the stream arriving from left to right.

$$AAAB\underline{B}BC$$

$\alpha = \text{A}$

$\text{c} = 1$

Majority algorithm - wrong answer

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
       if c == 0
            α = j
            c = 1
       elif j == α
            c = c + 1
       else
            c = c - 1
```

Consider the stream arriving from left to right.

$$AAABB\underline{B}C$$

$\alpha = \mathsf{A}$
c $= 0$

Majority algorithm - wrong answer

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
        if c == 0
            α = j
            c = 1
        elif j == α
            c = c + 1
        else
            c = c - 1
```

Consider the stream arriving from left to right.

$$AAABBB\underline{C}$$

$\alpha = \mathrm{C}$
c $= 1$

Time and space of the majority algorithm

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
  if c == 0
        α = j
        c = 1
  elif j == α
        c = c + 1
  else
        c = c - 1
```

Running time: At most $2m$ comparisons

$O(m)$ overall.

Time and space of the majority algorithm

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
  if c == 0
      α = j
      c = 1
  elif j == α
      c = c + 1
  else
      c = c - 1
```

Running time: At most $2m$ comparisons

$O(m)$ overall.

Space usage: One item and one integer

$O(\log n + \log m)$ bits overall

Correctness of the majority algorithm

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
  if c == 0
        α = j
        c = 1
  elif j == α
        c = c + 1
  else
        c = c - 1
```

If there is a majority item, it is reported.

Correctness of the majority algorithm

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
  if c == 0
      α = j
      c = 1
  elif j == α
      c = c + 1
  else
      c = c - 1
```

If there is a majority item, it is reported.

Let α^* be the final value of α

Correctness of the majority algorithm

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
  if c == 0
      α = j
      c = 1
  elif j == α
      c = c + 1
  else
      c = c - 1
```

If there is a majority item, it is reported.

Let α^* be the final value of α

Run through input from left to right.

Correctness of the majority algorithm

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
  if c == 0
      α = j
      c = 1
  elif j == α
      c = c + 1
  else
      c = c - 1
```

If there is a majority item, it is reported.

Let α^* be the final value of α

Run through input from left to right.

Let $c' = c$ if $\alpha = \alpha^*$ and $-c$ otherwise.

Correctness of the majority algorithm

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
  if c == 0
      α = j
      c = 1
  elif j == α
      c = c + 1
  else
      c = c - 1
```

If there is a majority item, it is reported.

Let α^* be the final value of α

Run through input from left to right.

Let $c' = c$ if $\alpha = \alpha^*$ and $-c$ otherwise.

Every occurrence of α^* increases c' by one.

Correctness of the majority algorithm

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
  if c == 0
      α = j
      c = 1
  elif j == α
      c = c + 1
  else
      c = c - 1
```

If there is a majority item, it is reported.

Let α^* be the final value of α

Run through input from left to right.

Let $c' = c$ if $\alpha = \alpha^*$ and $-c$ otherwise.

Every occurrence of α^* increases c' by one.

Every occurrence that is not α^* either increases or decreases c' by 1.

Correctness of the majority algorithm

```
MAJORITY(j)

initialise item α
initialise counter c = 0
Repeat for each j
  if c == 0
      α = j
      c = 1
  elif j == α
      c = c + 1
  else
      c = c - 1
```

If there is a majority item, it is reported.

Let α^* be the final value of α

Run through input from left to right.

Let $c' = c$ if $\alpha = \alpha^*$ and $-c$ otherwise.

Every occurrence of α^* increases c' by one.

Every occurrence that is not α^* either increases or decreases c' by 1.

If α^* is in the majority, there are more increases than decreases and so $c' = c$ which implies $\alpha = \alpha^*$. $\square$

Correctness of the majority algorithm

$\underline{A}AACCBBCCCBCC$

$\alpha = A$

$c = 1$

$c' = -1$

If there is a majority item, it is reported.

$A, A, A, C, C, B, B, C, C, C, B, C, C$
At termination $c = 3$, $\alpha^* = C$

Run through input from left to right.

Let $c' = c$ if $\alpha = \alpha^*$ and $-c$ otherwise.

Every occurence of α^* increases c' by one.

Every occurence that is not α^* either increases or decreases c' by 1.

If α^* is in the majority, more increases than decreases!

Correctness of the majority algorithm

$A\underline{A}ACCBBCCCBCC$

$\alpha = A$

$c = 2$

$c' = -2$

If there is a majority item, it is reported.

$A, A, A, C, C, B, B, C, C, C, B, C, C$
At termination $c = 3$, $\alpha^* = C$

Run through input from left to right.

Let $c' = c$ if $\alpha = \alpha^*$ and $-c$ otherwise.

Every occurence of α^* increases c' by one.

Every occurence that is not α^* either increases or decreases c' by 1.

If α^* is in the majority, more increases than decreases!

Correctness of the majority algorithm

$AA\underline{A}CCBBCCCBCC$

$\alpha = A$

$c = 3$

$c' = -3$

If there is a majority item, it is reported.

$A, A, A, C, C, B, B, C, C, C, B, C, C$
At termination $c = 3$, $\alpha^* = C$

Run through input from left to right.

Let $c' = c$ if $\alpha = \alpha^*$ and $-c$ otherwise.

Every occurence of α^* increases c' by one.

Every occurence that is not α^* either increases or decreases c' by 1.

If α^* is in the majority, more increases than decreases!

Correctness of the majority algorithm

$AAA\underline{C}CBBCCCBCC$

$\alpha = A$

$c = 2$

$c' = -2$

If there is a majority item, it is reported.

$A, A, A, C, C, B, B, C, C, C, B, C, C$
At termination $c = 3$, $\alpha^* = C$

Run through input from left to right.

Let $c' = c$ if $\alpha = \alpha^*$ and $-c$ otherwise.

Every occurence of α^* increases c' by one.

Every occurence that is not α^* either increases or decreases c' by 1.

If α^* is in the majority, more increases than decreases!

Correctness of the majority algorithm

$AAAC\underline{C}BBCCCBCC$

$\alpha = A$

$c = 1$

$c' = -1$

If there is a majority item, it is reported.

$$\boxed{\begin{array}{l} A, A, A, C, C, B, B, C, C, C, B, C, C \\ \text{At termination } c = 3,\ \alpha^* = C \end{array}}$$

Run through input from left to right.

Let $c' = c$ if $\alpha = \alpha^*$ and $-c$ otherwise.

Every occurence of α^* increases c' by one.

Every occurence that is not α^* either increases or decreases c' by 1.

If α^* is in the majority, more increases than decreases!

Correctness of the majority algorithm

$AAACC\underline{B}BCCCBCC$

$\alpha = A$

$c = 0$

$c' = 0$

If there is a majority item, it is reported.

$A, A, A, C, C, B, B, C, C, C, B, C, C$
At termination $c = 3$, $\alpha^* = C$

Run through input from left to right.

Let $c' = c$ if $\alpha = \alpha^*$ and $-c$ otherwise.

Every occurence of α^* increases c' by one.

Every occurence that is not α^* either increases or decreases c' by 1.

If α^* is in the majority, more increases than decreases!

Correctness of the majority algorithm

$AAACCB\underline{B}CCCBCC$

$\alpha = B$

$c = 1$

$c' = -1$

If there is a majority item, it is reported.

$A, A, A, C, C, B, B, C, C, C, B, C, C$
At termination $c = 3$, $\alpha^* = C$

Run through input from left to right.

Let $c' = c$ if $\alpha = \alpha^*$ and $-c$ otherwise.

Every occurence of α^* increases c' by one.

Every occurence that is not α^* either increases or decreases c' by 1.

If α^* is in the majority, more increases than decreases!

Correctness of the majority algorithm

If there is a majority item, it is reported.

$A, A, A, C, C, B, B, C, C, C, B, C, C$
At termination $c = 3$, $\alpha^* = C$

$AAACCBB\underline{C}CCBCC$

$\alpha = B$

$c = 0$

$c' = 0$

Run through input from left to right.

Let $c' = c$ if $\alpha = \alpha^*$ and $-c$ otherwise.

Every occurence of α^* increases c' by one.

Every occurence that is not α^* either increases or decreases c' by 1.

If α^* is in the majority, more increases than decreases!

Correctness of the majority algorithm

$AAACCBBC\underline{C}CBCC$

$\alpha = C$

$c = 1$

$c' = 1$

If there is a majority item, it is reported.

$\boxed{\begin{array}{l} A, A, A, C, C, B, B, C, C, C, B, C, C \\ \text{At termination } c = 3,\ \alpha^* = C \end{array}}$

Run through input from left to right.

Let $c' = c$ if $\alpha = \alpha^*$ and $-c$ otherwise.

Every occurence of α^* increases c' by one.

Every occurence that is not α^* either increases or decreases c' by 1.

If α^* is in the majority, more increases than decreases!

Correctness of the majority algorithm

$AAACCBBCC\underline{C}BCC$

$\alpha = C$

$c = 2$

$c' = 2$

If there is a majority item, it is reported.

$A, A, A, C, C, B, B, C, C, C, B, C, C$
At termination $c = 3$, $\alpha^* = C$

Run through input from left to right.

Let $c' = c$ if $\alpha = \alpha^*$ and $-c$ otherwise.

Every occurence of α^* increases c' by one.

Every occurence that is not α^* either increases or decreases c' by 1.

If α^* is in the majority, more increases than decreases!

Correctness of the majority algorithm

If there is a majority item, it is reported.

$A, A, A, C, C, B, B, C, C, C, B, C, C$
At termination $c = 3$, $\alpha^* = C$

$AAACCBBCCC\underline{B}CC$

$\alpha = C$

$c = 1$

$c' = 1$

Run through input from left to right.

Let $c' = c$ if $\alpha = \alpha^*$ and $-c$ otherwise.

Every occurence of α^* increases c' by one.

Every occurence that is not α^* either increases or decreases c' by 1.

If α^* is in the majority, more increases than decreases!

Correctness of the majority algorithm

$AAACCBBCCCB\underline{C}C$

$\alpha = C$

$c = 2$

$c' = 2$

If there is a majority item, it is reported.

$A, A, A, C, C, B, B, C, C, C, B, C, C$
At termination $c = 3$, $\alpha^* = C$

Run through input from left to right.

Let $c' = c$ if $\alpha = \alpha^*$ and $-c$ otherwise.

Every occurence of α^* increases c' by one.

Every occurence that is not α^* either increases or decreases c' by 1.

If α^* is in the majority, more increases than decreases!

Correctness of the majority algorithm

$AAACCBBCCCBC\underline{C}$

$\alpha = C$

$c = 3$

$c' = 3$

If there is a majority item, it is reported.

$A, A, A, C, C, B, B, C, C, C, B, C, C$
At termination $c = 3$, $\alpha^* = C$

Run through input from left to right.

Let $c' = c$ if $\alpha = \alpha^*$ and $-c$ otherwise.

Every occurence of α^* increases c' by one.

Every occurence that is not α^* either increases or decreases c' by 1.

If α^* is in the majority, more increases than decreases!

Misra-Gries - A generalisation of Majority

Given k, which elements (if any) appear more than m/k times?

```
MISRA-GRIES(a_1, a_2, ..., a_m)

set A = ∅
For each i
 if a_i ∈ A
  f̃_{a_i} = f̃_{a_i} + 1
 else
  if |A| < k − 1
   add (a_i, 1) to A
  else
   for (a_i, f̃_{a_i}) ∈ A
    f̃_{a_i} = f̃_{a_i} − 1
    if f̃_{a_i} = 0
     remove(a_i, f̃_{a_i}) from A
```

Misra-Gries - A generalisation of Majority

Given k, which elements (if any) appear more than m/k times?

```
MISRA-GRIES(a_1, a_2, ..., a_m)

set A = ∅
For each i
  if a_i ∈ A
    f̃_{a_i} = f̃_{a_i} + 1
  else
    if |A| < k − 1
      add (a_i, 1) to A
    else
      for (a_i, f̃_{a_i}) ∈ A
        f̃_{a_i} = f̃_{a_i} − 1
        if f̃_{a_i} = 0
          remove(a_i, f̃_{a_i}) from A
```

$\tilde{f}_{a_i}$ is the estimate for the frequency of token a_i

Misra-Gries - A generalisation of Majority

Given k, which elements (if any) appear more than m/k times?

```
MISRA-GRIES(a_1, a_2, ..., a_m)

set A = ∅
For each i
 if a_i ∈ A
  f̃_{a_i} = f̃_{a_i} + 1
 else
  if |A| < k − 1
   add (a_i, 1) to A
  else
   for (a_i, f̃_{a_i}) ∈ A
    f̃_{a_i} = f̃_{a_i} − 1
    if f̃_{a_i} = 0
     remove(a_i, f̃_{a_i}) from A
```

$\tilde{f}_{a_i}$ is the estimate for the frequency of token a_i

- Returns at most $k-1$ pairs $(v, \tilde{f}_v)$

Misra-Gries - A generalisation of Majority

Given k, which elements (if any) appear more than m/k times?

```
MISRA-GRIES(a_1, a_2, ..., a_m)

set A = ∅
For each i
  if a_i ∈ A
    f̃_{a_i} = f̃_{a_i} + 1
  else
    if |A| < k − 1
      add (a_i, 1) to A
    else
      for (a_i, f̃_{a_i}) ∈ A
        f̃_{a_i} = f̃_{a_i} − 1
        if f̃_{a_i} = 0
          remove(a_i, f̃_{a_i}) from A
```

$\tilde{f}_{a_i}$ is the estimate for the frequency of token a_i

- Returns at most $k - 1$ pairs $(v, \tilde{f}_v)$
- For every $(v, \tilde{f}_v) \in A$ where the true frequency is f_v,

$$f_v - \frac{m}{k} \leq \tilde{f}_v \leq f_v$$

Misra-Gries - A generalisation of Majority

Given k, which elements (if any) appear more than m/k times?

```
MISRA-GRIES(a_1, a_2, ..., a_m)

set A = ∅
For each i
  if a_i ∈ A
    f̃_{a_i} = f̃_{a_i} + 1
  else
   if |A| < k − 1
    add (a_i, 1) to A
   else
    for (a_i, f̃_{a_i}) ∈ A
     f̃_{a_i} = f̃_{a_i} − 1
     if f̃_{a_i} = 0
      remove(a_i, f̃_{a_i}) from A
```

$\tilde{f}_{a_i}$ is the estimate for the frequency of token a_i

- Returns at most $k-1$ pairs $(v, \tilde{f}_v)$
- For every $(v, \tilde{f}_v) \in A$ where the true frequency is f_v,

$$f_v - \frac{m}{k} \le \tilde{f}_v \le f_v$$

- Every $1/k$-heavy hitter is found.

Misra-Gries - A generalisation of Majority

Given k, which elements (if any) appear more than m/k times?

```
MISRA-GRIES(a_1, a_2, ..., a_m)

set A = ∅
For each i
  if a_i ∈ A
    f̃_{a_i} = f̃_{a_i} + 1
  else
    if |A| < k − 1
      add (a_i, 1) to A
    else
      for (a_i, f̃_{a_i}) ∈ A
        f̃_{a_i} = f̃_{a_i} − 1
        if f̃_{a_i} = 0
          remove(a_i, f̃_{a_i}) from A
```

$\tilde{f}_{a_i}$ is the estimate for the frequency of token a_i

- Returns at most $k-1$ pairs $(v, \tilde{f}_v)$
- For every $(v, \tilde{f}_v) \in A$ where the true frequency is f_v,

$$f_v - \frac{m}{k} \leq \tilde{f}_v \leq f_v$$

- Every $1/k$-heavy hitter is found.
- Some non-heaver hitters might be reported.

Misra-Gries - A generalisation of Majority

Given k, which elements (if any) appear more than m/k times?

```
MISRA-GRIES(a_1, a_2, ..., a_m)

set A = ∅
For each i
  if a_i ∈ A
    f̃_{a_i} = f̃_{a_i} + 1
  else
    if |A| < k − 1
      add (a_i, 1) to A
    else
      for (a_i, f̃_{a_i}) ∈ A
        f̃_{a_i} = f̃_{a_i} − 1
        if f̃_{a_i} = 0
          remove(a_i, f̃_{a_i}) from A
```

$\tilde{f}_{a_i}$ is the estimate for the frequency of token a_i

- Returns at most $k - 1$ pairs $(v, \tilde{f}_v)$
- For every $(v, \tilde{f}_v) \in A$ where the true frequency is f_v,

$$f_v - \frac{m}{k} \leq \tilde{f}_v \leq f_v$$

- Every $1/k$-heavy hitter is found.
- Some non-heaver hitters might be reported.
- Second pass may be needed

Misra-Gries - Worked example

Given k, which elements (if any) appear more than m/k times?

```
MISRA-GRIES(a_1, a_2, ..., a_m)

set A = ∅
For each i
 if a_i ∈ A
  f̃_{a_i} = f̃_{a_i} + 1
 else
  if |A| < k − 1
   add (a_i, 1) to A
  else
   for (a_i, f̃_{a_i}) ∈ A
    f̃_{a_i} = f̃_{a_i} − 1
    if f̃_{a_i} = 0
     remove(a_i, f̃_{a_i}) from A
```

Stream: $ACABACBB$
$m = 8, k = 3.$

Misra-Gries - Worked example

Given k, which elements (if any) appear more than m/k times?

MISRA-GRIES($a_1, a_2, \ldots, a_m$)

```
set A = ∅
For each i
  if a_i ∈ A
    f̃_{a_i} = f̃_{a_i} + 1
  else
    if |A| < k − 1
      add (a_i, 1) to A
    else
      for (a_i, f̃_{a_i}) ∈ A
        f̃_{a_i} = f̃_{a_i} − 1
        if f̃_{a_i} = 0
          remove(a_i, f̃_{a_i}) from A
```

Stream: $ACABACBB$
$m = 8, k = 3.$

$A \rightarrow \tilde{f}_A = 1$

Misra-Gries - Worked example

Given k, which elements (if any) appear more than m/k times?

```
MISRA-GRIES(a_1, a_2, ..., a_m)

set A = ∅
For each i
  if a_i ∈ A
    f̃_{a_i} = f̃_{a_i} + 1
  else
    if |A| < k − 1
      add (a_i, 1) to A
    else
      for (a_i, f̃_{a_i}) ∈ A
        f̃_{a_i} = f̃_{a_i} − 1
        if f̃_{a_i} = 0
          remove(a_i, f̃_{a_i}) from A
```

Stream: $ACABACBB$
$m = 8, k = 3.$

$A \rightarrow \tilde{f}_A = 1$

$C \rightarrow \tilde{f}_A = 1, \tilde{f}_C = 1$

Misra-Gries - Worked example

Given k, which elements (if any) appear more than m/k times?

```
MISRA-GRIES(a_1, a_2, ..., a_m)

set A = ∅
For each i
  if a_i ∈ A
    f~_{a_i} = f~_{a_i} + 1
  else
    if |A| < k - 1
      add (a_i, 1) to A
    else
      for (a_i, f~_{a_i}) ∈ A
        f~_{a_i} = f~_{a_i} - 1
        if f~_{a_i} = 0
          remove(a_i, f~_{a_i}) from A
```

Stream: *ACABACBB*
$m = 8, k = 3.$

$A \to \tilde{f}_A = 1$

$C \to \tilde{f}_A = 1, \tilde{f}_C = 1$

$A \to \tilde{f}_A = 2, \tilde{f}_C = 1$

Misra-Gries - Worked example

Given k, which elements (if any) appear more than m/k times?

```
MISRA-GRIES(a_1, a_2, ..., a_m)

set A = ∅
For each i
 if a_i ∈ A
  f̃_{a_i} = f̃_{a_i} + 1
 else
  if |A| < k − 1
   add (a_i, 1) to A
  else
   for (a_i, f̃_{a_i}) ∈ A
    f̃_{a_i} = f̃_{a_i} − 1
    if f̃_{a_i} = 0
     remove(a_i, f̃_{a_i}) from A
```

Stream: *ACABACBB*
$m = 8, k = 3.$

$A \to \tilde{f}_A = 1$

$C \to \tilde{f}_A = 1, \tilde{f}_C = 1$

$A \to \tilde{f}_A = 2, \tilde{f}_C = 1$

$B \to \tilde{f}_A = 1$

Misra-Gries - Worked example

Given k, which elements (if any) appear more than m/k times?

```
MISRA-GRIES(a_1, a_2, ..., a_m)

set A = ∅
For each i
  if a_i ∈ A
   f~_{a_i} = f~_{a_i} + 1
  else
   if |A| < k - 1
    add (a_i, 1) to A
   else
    for (a_i, f~_{a_i}) ∈ A
     f~_{a_i} = f~_{a_i} - 1
     if f~_{a_i} = 0
      remove(a_i, f~_{a_i}) from A
```

Stream: $ACABACBB$
$m = 8, k = 3.$

$A \to \tilde{f}_A = 1$

$C \to \tilde{f}_A = 1, \tilde{f}_C = 1$

$A \to \tilde{f}_A = 2, \tilde{f}_C = 1$

$B \to \tilde{f}_A = 1$

$A \to \tilde{f}_A = 2$

Misra-Gries - Worked example

Given k, which elements (if any) appear more than m/k times?

```
MISRA-GRIES(a_1, a_2, ..., a_m)

set A = ∅
For each i
  if a_i ∈ A
    f̃_{a_i} = f̃_{a_i} + 1
  else
    if |A| < k − 1
      add (a_i, 1) to A
    else
      for (a_i, f̃_{a_i}) ∈ A
        f̃_{a_i} = f̃_{a_i} − 1
        if f̃_{a_i} = 0
          remove(a_i, f̃_{a_i}) from A
```

Stream: *ACABACBB*
$m = 8, k = 3.$

$A \rightarrow \tilde{f}_A = 1$

$C \rightarrow \tilde{f}_A = 1, \tilde{f}_C = 1$

$A \rightarrow \tilde{f}_A = 2, \tilde{f}_C = 1$

$B \rightarrow \tilde{f}_A = 1$

$A \rightarrow \tilde{f}_A = 2$

$C \rightarrow \tilde{f}_A = 2, \tilde{f}_C = 1$

Misra-Gries - Worked example

Given k, which elements (if any) appear more than m/k times?

```
MISRA-GRIES(a_1, a_2, ..., a_m)

set A = ∅
For each i
  if a_i ∈ A
    f̃_{a_i} = f̃_{a_i} + 1
  else
    if |A| < k − 1
      add (a_i, 1) to A
    else
      for (a_i, f̃_{a_i}) ∈ A
        f̃_{a_i} = f̃_{a_i} − 1
        if f̃_{a_i} = 0
          remove(a_i, f̃_{a_i}) from A
```

Stream: $ACABACBB$
$m = 8, k = 3.$

$A \to \tilde{f}_A = 1$

$C \to \tilde{f}_A = 1, \tilde{f}_C = 1$

$A \to \tilde{f}_A = 2, \tilde{f}_C = 1$

$B \to \tilde{f}_A = 1$

$A \to \tilde{f}_A = 2$

$C \to \tilde{f}_A = 2, \tilde{f}_C = 1$

$B \to \tilde{f}_A = 1$

Misra-Gries - Worked example

Given k, which elements (if any) appear more than m/k times?

```
MISRA-GRIES(a_1, a_2, ..., a_m)

set A = ∅
For each i
  if a_i ∈ A
   f̃_{a_i} = f̃_{a_i} + 1
  else
   if |A| < k − 1
    add (a_i, 1) to A
   else
    for (a_i, f̃_{a_i}) ∈ A
     f̃_{a_i} = f̃_{a_i} − 1
     if f̃_{a_i} = 0
      remove(a_i, f̃_{a_i}) from A
```

Stream: $ACABACBB$
$m = 8, k = 3$.

$A \to \tilde{f}_A = 1$

$C \to \tilde{f}_A = 1, \tilde{f}_C = 1$

$A \to \tilde{f}_A = 2, \tilde{f}_C = 1$

$B \to \tilde{f}_A = 1$

$A \to \tilde{f}_A = 2$

$C \to \tilde{f}_A = 2, \tilde{f}_C = 1$

$B \to \tilde{f}_A = 1$

$B \to \tilde{f}_A = 1, \tilde{f}_B = 1$

Misra-Gries - Space/Time

Given k, which elements (if any) appear more than m/k times?

```
MISRA-GRIES(a_1, a_2, ..., a_m)

set A = ∅
For each i
  if a_i ∈ A
    f̃_{a_i} = f̃_{a_i} + 1
  else
    if |A| < k − 1
      add (a_i, 1) to A
    else
      for (a_i, f̃_{a_i}) ∈ A
        f̃_{a_i} = f̃_{a_i} − 1
        if f̃_{a_i} = 0
          remove(a_i, f̃_{a_i}) from A
```

- Space: $k-1$ pairs stored in total

Misra-Gries - Space/Time

Given k, which elements (if any) appear more than m/k times?

```
MISRA-GRIES(a_1, a_2, ..., a_m)

set A = ∅
For each i
  if a_i ∈ A
    f̃_{a_i} = f̃_{a_i} + 1
  else
    if |A| < k − 1
      add (a_i, 1) to A
    else
      for (a_i, f̃_{a_i}) ∈ A
        f̃_{a_i} = f̃_{a_i} − 1
        if f̃_{a_i} = 0
          remove(a_i, f̃_{a_i}) from A
```

- Space: $k-1$ pairs stored in total
- $O(k(\log m + \log n))$ bits space.

Misra-Gries - Space/Time

Given k, which elements (if any) appear more than m/k times?

```
MISRA-GRIES(a_1, a_2, ..., a_m)

set A = ∅
For each i
  if a_i ∈ A
    f̃_{a_i} = f̃_{a_i} + 1
  else
    if |A| < k − 1
      add (a_i, 1) to A
    else
      for (a_i, f̃_{a_i}) ∈ A
        f̃_{a_i} = f̃_{a_i} − 1
        if f̃_{a_i} = 0
          remove(a_i, f̃_{a_i}) from A
```

- Space: $k-1$ pairs stored in total
- $O(k(\log m + \log n))$ bits space.

Misra-Gries - Space/Time

Given k, which elements (if any) appear more than m/k times?

```
MISRA-GRIES(a_1, a_2, ..., a_m)

set A = ∅
For each i
 if a_i ∈ A
  f̃_{a_i} = f̃_{a_i} + 1
 else
  if |A| < k − 1
   add (a_i, 1) to A
  else
   for (a_i, f̃_{a_i}) ∈ A
    f̃_{a_i} = f̃_{a_i} − 1
    if f̃_{a_i} = 0
     remove(a_i, f̃_{a_i}) from A
```

- Space: $k-1$ pairs stored in total
- $O(k(\log m + \log n))$ bits space.

- Running time depends on data structure used.

Misra-Gries - Space/Time

Given k, which elements (if any) appear more than m/k times?

```
MISRA-GRIES(a_1, a_2, ..., a_m)

set A = ∅
For each i
 if a_i ∈ A
  f̃_{a_i} = f̃_{a_i} + 1
 else
  if |A| < k − 1
   add (a_i, 1) to A
  else
   for (a_i, f̃_{a_i}) ∈ A
    f̃_{a_i} = f̃_{a_i} − 1
    if f̃_{a_i} = 0
     remove(a_i, f̃_{a_i}) from A
```

- Space: $k - 1$ pairs stored in total
- $O(k(\log m + \log n))$ bits space.

- Running time depends on data structure used.
- Balanced binary search tree $O(\log n)$ time per operation.

Misra-Gries - Space/Time

Given k, which elements (if any) appear more than m/k times?

```
MISRA-GRIES(a_1, a_2, ..., a_m)

set A = ∅
For each i
  if a_i ∈ A
    f̃_{a_i} = f̃_{a_i} + 1
  else
    if |A| < k − 1
      add (a_i, 1) to A
    else
      for (a_i, f̃_{a_i}) ∈ A
        f̃_{a_i} = f̃_{a_i} − 1
        if f̃_{a_i} = 0
          remove(a_i, f̃_{a_i}) from A
```

- Space: $k-1$ pairs stored in total
- $O(k(\log m + \log n))$ bits space.

- Running time depends on data structure used.
- Balanced binary search tree $O(\log n)$ time per operation.
- We can only decrement (or remove) if we previously incremented. Therefore $O(m)$ operations.

Misra-Gries - Space/Time

Given k, which elements (if any) appear more than m/k times?

```
MISRA-GRIES(a_1, a_2, ..., a_m)

set A = ∅
For each i
  if a_i ∈ A
    f̃_{a_i} = f̃_{a_i} + 1
  else
    if |A| < k − 1
      add (a_i, 1) to A
    else
      for (a_i, f̃_{a_i}) ∈ A
        f̃_{a_i} = f̃_{a_i} − 1
        if f̃_{a_i} = 0
          remove(a_i, f̃_{a_i}) from A
```

- Space: $k-1$ pairs stored in total
- $O(k(\log m + \log n))$ bits space.

- Running time depends on data structure used.
- Balanced binary search tree $O(\log n)$ time per operation.
- We can only decrement (or remove) if we previously incremented. Therefore $O(m)$ operations.
- $O(m(\log n))$ time overall.

Modified Misra-Gries - Analysis

Let's look at a less efficient version for the analysis.

```
MODIFIED-MG(a_1, a_2, ..., a_m)

set A = empty multiset
For each i
 if a_i ∈ A
  add a copy of a_i to A
 else
  if |supp(A)| < k − 1
   add a_i to A
  else
   add and then delete a_i
   delete one copy of each
                  item in A
```

Modified Misra-Gries - Analysis

Let's look at a less efficient version for the analysis.

```
MODIFIED-MG(a_1, a_2, ..., a_m)

set A = empty multiset
For each i
 if a_i ∈ A
  add a copy of a_i to A
 else
  if |supp(A)| < k − 1
   add a_i to A
  else
   add and then delete a_i
   delete one copy of each
                 item in A
```

- $|\text{supp}(A)|$ is the number of distinct tokens in A.

Modified Misra-Gries - Analysis

Let's look at a less efficient version for the analysis.

```
MODIFIED-MG(a_1, a_2, ..., a_m)

set A = empty multiset
For each i
 if a_i ∈ A
  add a copy of a_i to A
 else
  if |supp(A)| < k − 1
   add a_i to A
  else
   add and then delete a_i
   delete one copy of each
                  item in A
```

- $|\mathrm{supp}(A)|$ is the number of distinct tokens in A.
- Identical except for space usage.

Modified Misra-Gries - Analysis

Let's look at a less efficient version for the analysis.

```
MODIFIED-MG(a_1, a_2, ..., a_m)

set A = empty multiset
For each i
 if a_i ∈ A
  add a copy of a_i to A
 else
  if |supp(A)| < k − 1
   add a_i to A
  else
   add and then delete a_i
   delete one copy of each
                  item in A
```

- $|\mathrm{supp}(A)|$ is the number of distinct tokens in A.
- Identical except for space usage.
- Items are deleted in groups of k.

Modified Misra-Gries - Analysis

Let's look at a less efficient version for the analysis.

```
MODIFIED-MG(a_1, a_2, ..., a_m)

set A = empty multiset
For each i
 if a_i ∈ A
  add a copy of a_i to A
 else
  if |supp(A)| < k − 1
   add a_i to A
  else
   add and then delete a_i
   delete one copy of each
                 item in A
```

- $|\text{supp}(A)|$ is the number of distinct tokens in A.
- Identical except for space usage.
- Items are deleted in groups of k.
- Each item can be deleted $\leq \frac{m}{k}$ times.

Misra-Gries - Analysis

Statement for Misra-Gries

For every $(v, \tilde{f}_v) \in A$ where the true frequency is f_v,

$$f_v - \frac{m}{k} \leq \tilde{f}_v \leq f_v$$

Misra-Gries - Analysis

Statement for Misra-Gries

For every $(v, \tilde{f}_v) \in A$ where the true frequency is f_v,

$$f_v - \frac{m}{k} \leq \tilde{f}_v \leq f_v$$

- Items are deleted in groups of k.

Misra-Gries - Analysis

Statement for Misra-Gries

For every $(v, \tilde{f}_v) \in A$ where the true frequency is f_v,

$$f_v - \frac{m}{k} \leq \tilde{f}_v \leq f_v$$

- Items are deleted in groups of k.
- Each item can therefore be deleted $\leq \frac{m}{k}$ times.

Misra-Gries - Analysis

Statement for Misra-Gries

For every $(v, \tilde{f}_v) \in A$ where the true frequency is f_v,

$$f_v - \frac{m}{k} \leq \tilde{f}_v \leq f_v$$

- Items are deleted in groups of k.
- Each item can therefore be deleted $\leq \frac{m}{k}$ times.
- Let $a(v)$ be the number of times v was seen without incrementing $\tilde{f}_v$. Let $b(v)$ be the number of times $\tilde{f}_v$ was decremented.

Misra-Gries - Analysis

Statement for Misra-Gries

For every $(v, \tilde{f}_v) \in A$ where the true frequency is f_v,

$$f_v - \frac{m}{k} \leq \tilde{f}_v \leq f_v$$

- Items are deleted in groups of k.
- Each item can therefore be deleted $\leq \frac{m}{k}$ times.
- Let $a(v)$ be the number of times v was seen without incrementing $\tilde{f}_v$. Let $b(v)$ be the number of times $\tilde{f}_v$ was decremented.
- We now have that $\tilde{f}_v = f_v - a(v) - b(v) = f_v - (a(v) + b(v))$.

Misra-Gries - Analysis

Statement for Misra-Gries

For every $(v, \tilde{f}_v) \in A$ where the true frequency is f_v,

$$f_v - \frac{m}{k} \leq \tilde{f}_v \leq f_v$$

- Items are deleted in groups of k.
- Each item can therefore be deleted $\leq \frac{m}{k}$ times.
- Let $a(v)$ be the number of times v was seen without incrementing $\tilde{f}_v$. Let $b(v)$ be the number of times $\tilde{f}_v$ was decremented.
- We now have that $\tilde{f}_v = f_v - a(v) - b(v) = f_v - (a(v) + b(v))$.
- The number of decrements equals $a(v) + b(v)$, therefore $a(v) + b(v) \leq \frac{m}{k}$.

Misra-Gries - Analysis

Statement for Misra-Gries

For every $(v, \tilde{f}_v) \in A$ where the true frequency is f_v,

$$f_v - \frac{m}{k} \leq \tilde{f}_v \leq f_v$$

- Items are deleted in groups of k.
- Each item can therefore be deleted $\leq \frac{m}{k}$ times.
- Let $a(v)$ be the number of times v was seen without incrementing $\tilde{f}_v$. Let $b(v)$ be the number of times $\tilde{f}_v$ was decremented.
- We now have that $\tilde{f}_v = f_v - a(v) - b(v) = f_v - (a(v) + b(v))$.
- The number of decrements equals $a(v) + b(v)$, therefore $a(v) + b(v) \leq \frac{m}{k}$.
- Hence $f_v - \frac{m}{k} \leq \tilde{f}_v$.

Summary

Misra-Gries

The Misra-Gries algorithm uses $O(\frac{1}{\epsilon}(\log m + \log n))$ bits of space to find a set of size at most $\lceil \frac{1}{\epsilon} \rceil$ that contains every item of frequency at least ϵm.

Summary

Misra-Gries

The Misra-Gries algorithm uses $O(\frac{1}{\epsilon}(\log m + \log n))$ bits of space to find a set of size at most $\lceil \frac{1}{\epsilon} \rceil$ that contains every item of frequency at least ϵm.

1. Majority uses $O(\log m + \log n)$ space and runs in linear time.

Summary

Misra-Gries

The Misra-Gries algorithm uses $O(\frac{1}{\epsilon}(\log m + \log n))$ bits of space to find a set of size at most $\lceil \frac{1}{\epsilon} \rceil$ that contains every item of frequency at least ϵm.

1. Majority uses $O(\log m + \log n)$ space and runs in linear time.
2. If there is an item that occurs more than $m/2$ times the Majority algorithm will output it.

Summary

Misra-Gries

The Misra-Gries algorithm uses $O(\frac{1}{\epsilon}(\log m + \log n))$ bits of space to find a set of size at most $\lceil \frac{1}{\epsilon} \rceil$ that contains every item of frequency at least ϵm.

1. Majority uses $O(\log m + \log n)$ space and runs in linear time.
2. If there is an item that occurs more than $m/2$ times the Majority algorithm will output it.
3. With a second pass we can check if there is a majority in the stream

Summary

Misra-Gries

The Misra-Gries algorithm uses $O(\frac{1}{\epsilon}(\log m + \log n))$ bits of space to find a set of size at most $\lceil \frac{1}{\epsilon} \rceil$ that contains every item of frequency at least ϵm.

1. Majority uses $O(\log m + \log n)$ space and runs in linear time.
2. If there is an item that occurs more than $m/2$ times the Majority algorithm will output it.
3. With a second pass we can check if there is a majority in the stream
4. Misra-Gries uses $O(k(\log m + \log n))$ space and runs in $O(m \log m)$ time.

Summary

MISRA-GRIES

The Misra-Gries algorithm uses $O(\frac{1}{\epsilon}(\log m + \log n))$ bits of space to find a set of size at most $\lceil \frac{1}{\epsilon} \rceil$ that contains every item of frequency at least ϵm.

1. MAJORITY uses $O(\log m + \log n)$ space and runs in linear time.
2. If there is an item that occurs more than $m/2$ times the MAJORITY algorithm will output it.
3. With a second pass we can check if there is a majority in the stream
4. MISRA-GRIES uses $O(k(\log m + \log n))$ space and runs in $O(m \log m)$ time.
5. It will output all tokens that occur more than m/k times but may output others as well.

Summary

Misra-Gries

The Misra-Gries algorithm uses $O(\frac{1}{\epsilon}(\log m + \log n))$ bits of space to find a set of size at most $\lceil \frac{1}{\epsilon} \rceil$ that contains every item of frequency at least ϵm.

1. Majority uses $O(\log m + \log n)$ space and runs in linear time.
2. If there is an item that occurs more than $m/2$ times the Majority algorithm will output it.
3. With a second pass we can check if there is a majority in the stream
4. Misra-Gries uses $O(k(\log m + \log n))$ space and runs in $O(m \log m)$ time.
5. It will output all tokens that occur more than m/k times but may output others as well.
6. With a second pass we can remove all the undesired tokens.